PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/455, 9/44 — A1

(11) International Publication Number: WO 97/14098

(43) International Publication Date: 17 April 1997 (17.04.97)

(21) International Application Number: PCT/US96/15947

(22) International Filing Date: 4 October 1996 (04.10.96)

(30) Priority Data:
08/541,020 11 October 1995 (11.10.95) US

(71) Applicant: CITRIX SYSTEMS, INC. [US/US]; Suite 700, 210 University Drive, Coral Springs, FL 33071 (US).

(72) Inventors: DISCAVAGE, Michael, J.; 5214 Northwest, 27 Court, Margate, FL 33063 (US). IACOBUCCI, Edward, E.; Apartment 302, 1921 Lyons Road, Coconut Creek, FL 33063 (US).

(74) Agent: TURANO, Thomas, A.; Testa, Hurwitz & Thibeault, L.L.P., High Street Tower, 125 High Street, Boston, MA 02110 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: A METHOD FOR PROVIDING USER GLOBAL OBJECT NAME SPACE IN A MULTI-USER OPERATING SYSTEM

(57) Abstract

A method, suitable for use in client/server system, which allows multiple copies of a single-user application to run simultaneously in a multi-user operating system without modification of the single-user program, by modifying existing operating system methods used for object name creation, lookup, and deletion. The method creates a user global context by labeling each instance of the single-user application with a user identifier (name) that defines a single-user name space in which each labeled object is only available to the named user. In addition, the single-user server process is allowed to impersonate the client for allowing the server to access the named resources of the single-user name space. A coexisting system global context is also created by marking system global named resources.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

A METHOD FOR PROVIDING USER GLOBAL OBJECT NAME SPACE IN A MULTI-USER OPERATING SYSTEM

FIELD OF INVENTION

The invention relates to multi-user distributed process systems having a master process and a plurality of service processes and a method for adapting applications written for a single user environment for use in the multi-user system without recompiling.

BACKGROUND TO THE INVENTION

The distributed process system described is a multi-user and multi-tasking system. Applications written for a multi-user environment typically assume that more than one copy of the application may be active at the same time. Multi-tasking operating systems (Windows NT and Unix for example) provide the capability to create various types of shared objects for providing interprocess communications and synchronization. By associating a name with an object, it allows for one process to create an object and for another process to make use of it. Synchronization ensures that only one process has control of a resource at a time. Resources include objects such as global variables, common data memory, file handles, and shared object handles.

Shared objects used for synchronization and interprocess communication include:

Semaphores and event objects for coordinating threads; Mutexes, a mutually exclusive semaphore allowing a single thread access at a time;

Named Pipes for duplexed interprocess and interprocessor connection;

Message Queues for one way, many to one communications; and

Shared Memory for use by all processes that have authorized access.

If an application was written for a single user for running under a single user operating system, such as Windows NT (produced by Microsoft Corp. of Redmond, Washington), and then is run in a multi-user environment under a compatible multi-user operating system, such as WinFrame™ (produced by Citrix Systems, Inc. of Coral Springs, Florida), in a multi-user environment, it is possible for name collisions to occur if more than one copy of the same application is executed at the same time. The application would have to be modified and recompiled in order for it to execute reliably in the multi-user environment.

The present invention modifies the existing methods used for object name creation, look-up, and deletion in a multi-user operating system so that multiple copies of a single user application are able to run simultaneously.

SUMMARY OF THE INVENTION

A method is described for allowing a single-user application program to operate in a multi-user operating system without modification of the single-user program by modifying existing operating system methods used for object name creation,, look-up, and deletion, so that multiple copies of a single-user application program are able to run simultaneously. The method includes the following steps:

a) assigning a unique identifier to each user on the system and each of the user's applications, and attaching this same identifier to each instance of an object created by the user's applications, for the purpose of creating a distinct single user name space that is only accessible by the same single user; and

b) enabling a server process that is serving the application of the single user process to impersonate the single user process by assuming the identity of the single user process, for allowing the server process to access the single user name space.

In this manner, the server process assumes the role of the user, has access to the user's private name space and to all objects required for serving the user's application. The combination of user labeling and user impersonation allows multiple copies of a given application to run simultaneously even though the application was written for a single-user operating system.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention may be more fully understood from the detailed description given below and from the accompanying drawings of the preferred embodiments of the invention, which, however, should not be taken to limit the invention to the specific embodiment but are only for explanation and understanding.

Fig. 1 shows the relationship between a session manager and multiple application servers in a multi-user system.

Fig. 2 shows the architecture of an application server.

Fig. 3(a) shows an example of a prior art create-object call.

Fig. 3(b) shows an example of a prior art open-object call.

Fig. 3(c) shows an example of a prior art delete-object call.

Fig. 4(a) shows a create-object call example in a system with user global name space.

Fig. 4(b) shows an open-object call example in a system with user global name space.

Fig. 4(c) shows a delete-object call example in a system with user global name space.

Fig. 5 describes the context of named resources that depend on the .EXE/.DLL overrides and on the API override values.

Fig. 6 is a flow diagram for creating a user and system global context.

DETAILED DESCRIPTION OF THE INVENTION

Two important operating system (OS) features for use in a client-server distributed process system are multi-tasking and multi-user operation.

Multi-tasking is the ability to run more than one program at a time by dividing each task into multiple "threads" or subtasks so that distinct tasks can share a single processor by multiplexing the threads of each task. In this manner, each distinct task is given access to the processor on a shared basis. Priority based scheduling manages the processor time each task or process is allocated.

In a OS like Windows NT, each process is divided into multiple threads for execution. The threads are run while the processes are never run. Each process may consist of one or more threads. Each thread has an assigned priority level that can be set independently.

A "process" is a single group of memory addresses in which some of the addresses are allocated to program steps while the rest are allocated for data required by the program steps. Typically, processes do not share memory with other processes because each process is structured as a single entity and its memory space is protected from unauthorized access by other processes. Multiple threads are created within a process with each thread having access to all memory and other resources within the process.

Each thread has three possible states that include:

Running - when the thread is executing;

Ready - when thread is waiting for other threads to finish executing; and

Blocked - when the thread is waiting for some event to occur.

Because processes may have need to communicate with each other, a variety of methods are provided that include synchronization and interprocess communication objects such as:

Event objects for notifying a waiting thread that an event has occurred;

Mutex (mutual exclusion) objects for use between processes to ensure only one process has access at a time;

Semaphore objects for use when a counter is required for limiting the number of threads that can use a given resource;

Named Pipe objects are one way or duplex stream connections between two processes; and

Communication Port objects are message connections between two processes.

By associating a name with each of these objects, the objects become capable of being shared between processes. An additional named object is named shared memory that allows one process to create the named object and another to make use of it in a multi-tasking system.

Fig. 1 shows the architecture of a multi-user application server system 100 resident in a host system and includes a session manager 101 and a number of application servers 200. Each application server 200 interfaces to a client (user) workstation for running the client's application in the host system by inputting data from and providing processed data and display data to the associated client workstation. The host system contains common resources that are shared amongst the numerous client workstations through the associated application server 200.

Fig. 2 shows the architecture of a commercial application server 200, manufactured by Citrix Systems, Inc., Coral Springs, Florida. The application server 200 includes subsystem 210 for the management of the associated client workstation for which application services are to be provided. Subsystem 210 includes a dynamically linkable library (DLL) and generates the control data-stream needed to control the workstation being accessed. WinStation Driver 10 interprets the data stream generated by either the Protocol Driver Stack 20, subsystem 210 or the client workstation. Protocol Driver Stack 20 is a communications layer for preparing data to be transmitted to or received from the workstation. Transport Driver 21 is a protocol driver for interfacing Application Server 200 to the Transport Service 300 being used by the workstation. Virtual Driver 60 runs a virtual channel protocol used for communicating application data to and from the user workstation.

In a multi-user multi-tasking system as shown in Fig. 1, applications must be written so that more than one copy of the application can run at the same time without object name collisions occurring. However, most applications written for a single user environment implicitly assume that only one copy of the application will be active in the system at a given time. Consequently, if more than one such application were to be run at the same time in a multi-user environment, there is a reasonable probability for name collisions to occur. This would make it impractical to use many existing single-user applications in a multi-user environment without modifying the application program.

For example, if a single user Windows NT operating system application that uses a named resource, such as a semaphore, were to attempt to run in a multi-user environment, it would initially attempt to create the named semaphore. However, if the same application had been previously started by another user, the attempt to create the named semaphore would fail. This

type o f application would have difficulty running multiple instances of itself in a single user environment. In order for it to run in a multi-user environment (e.g. under operating system WinFrame™), constructs must be included that localize global resources to the user level. The named resources that can run into this type of problem include: semaphores, named pipes, queues, communication ports, and named shared memory.

Figs. 3(a,b,c) are flow diagrams depicting the actions and results when two single-user processes (process A and process B) attempt to operate in a multi-user prior art environment. The first column (on the left) shows actions initiated by process A and the results while the second column shows the actions taken by process B and the results. The three flow diagrams are: Fig. 3(a) for creating an object named "XYZ"; Fig. 3(b) for opening an object named "XYZ"; and Fig. 3(c) for deleting an object named "XYZ".

In step 11 of Fig. 3(a), application process A calls an application program interface (API) in the host system's operating system (OS) kernel for creating an object named "XYZ". Then process B makes the same request in step 21. In step 12, object "XYZ" is created by the OS kernel as requested by process A and the name is stored. However, the request by process B in step 22 fails because the OS finds that the named object "XYZ" already exists. As a result, process A receives the required object handle in step 13, while process B receives an error status message.

In Fig. 3(b), both application processes A and B call an API to open the object named "XYZ". The OS conducts a search for object "XYZ" in steps 15 and 25. The two searches find the previously created object named "XYZ" and the object handle is returned to process A and B respectively in steps 16 and 26.

In Fig. 3(c) at step 17, process A calls an API for deleting an object named "XYZ". Assume that process B subsequently makes the same request in step 27. As a result, the request by process A causes a search for the object named "XYZ" in step 18, and results in the object being deleted in step 19. Because of the earlier request to delete by process A, the search for the object named "XYZ" in step 28 fails and step 29 returns an error status message.

The actions and results shown in Fig. 3(b) are not proper because process B gained access to object "XYZ" created by process A and not to the process named "XYZ" that was to be created by process B. Similarly, the actions and results shown in Fig. 3(c) may be undesirable because an object named "XYZ" could be deleted by either process A or B as determined by which process called for the deletion first.

In order to accommodate Windows NT, single user applications in a multi-user WinFrame™ operating environment, the following steps are taken:

(1) all application program interface (API) calls by a given user for these named resources are intercepted;

(2) a user identifier is added to the name before it is passed on in the API call; and

(3) all applications running on behalf of the given user will have the named requests for resources identically modified.

In this manner, these named resources are made "user global" which makes them shareable only within the given user's context.

Figs. 4(a,b,c) show the results of the above steps taken to accommodate single user applications in the multi-user system and the consequence of actions taken by process A on the left and process B on the right.

In Fig. 4(a) at step 30, process A calls an API in the OS kernel to create an object named "XYZ". The OS assigns ID(1) as a unique user identification (ID) associated with application process A in step 31. In step 32, the OS creates the object and saves the name "XYZ" together with the unique user ID(A). In step 33 the system returns the object handle to process A. Meanwhile, process B initiates similar actions in step 50 calling for the creation of an object named "XYZ" with the result that a unique user ID(B) is associated with the application process in step 51 and an object named "XYZ" is created in step 52 and the name is saved together with ID(B). The object handle for the object named "XYZ" associated with ID(B) is returned to process B. It should be noted that process A and B refer to their respective objects named "XYZ" using the same name but the OS clearly distinguishes them because of the associated user ID.

Fig. 4(b) is a flow diagram for opening an object by process A and B. In step 34, process A calls an API to open an object named "XYZ". In step 35 the unique ID that has been assigned to process A, ID(A), is retrieved by the OS and a search for the object named "XYZ" with the assigned unique user ID is made in step 36. If a match is found for both the name "XYZ" and ID(1), step 37 returns the handle for the desired object. Step 54 initiates a similar call from application process B to open an object named 5 "XYZ" and the OS assigns the unique user ID, ID(B), associated with process B and searches for object "XYZ" together with ID(B). Because of the unique ID assigned to each process, two same named objects can be separately supported by the OS. Consequently, step 56 searches and is able to find object "XYZ" belonging to process B. The proper object handle is returned to process B in step 57.

Fig. 4(c) is a flow diagram of deleting objects having the same name "XYZ". In step 38, process A calls an API to delete the object named "XYZ". Again, the OS retrieves the unique identifier ID(A) associated with process A in step 39 and proceeds to search for "XYZ" with matching identifier ID(A). When found, the object is deleted in step 41. Similarly, process B calls for deletion of object "XYZ" in step 58 and the following steps 59, 60 and 61 result in the deletion of object "XYZ" associated with ID(B).

Thus, by associating a unique ID with each named object, a single user application can be used in a multi-user system without modification by using the OS modifications described above.

Indiscriminately applying the user identifier to API calls can lead to problems. For example, if a multi-user application uses a system semaphore to serialize access to a protected resource, a "system global" context is needed rather than a user global context. Because the characteristics of the application making the API call are not readily available to the application program kernel, this potential problem is resolved by

(1) establishing user global as the default context,

(2) establishing a system global context,

(3) enabling an application to select between user global and system global context for a specific named object on an individual API basis by appending a context modifier to the name of the named object, and

(4) enabling the marking of specific application executable (.EXE) files or dynamically linkable library (.DLL) files so that all API calls from the executable or dynamically linked files are of the system global context.

In the WinFrame™ environment, multi-user applications and dynamic link libraries can intermix allocation of system global and user global named resources from within the same .EXE or .DLL. Because WinFrame™ requires that the resource type be specified at the end of the name string, accommodation of the named resource APIs require that the applications append the system global identifier to the name string.

Fig. 5 describes the context of named resources as a function of the user and system global modifiers applied to .EXE/.DLL and to the API call. If system global is not specified in the API call, the context is system global if .EXE/.DLL is marked system global. Otherwise, the context is user global. If the API call specifies system global, the context is system global independent of the .EXE/.DLL marking.

Because the client portion of an application in a distributed process client-server system generally resides at another location than the server, the server must verify the security or privilege level of the client. Also, if the client makes a request to access data in a file controlled by the server by use of an API call to open, read, or write the file, the server must have open/read/write privileges for that file. For example, if the file to be accessed by the application server is owned by the client as a user global file, access would be limited to the client. The concept of "impersonation" provides a means for resolving this dilemma.

Impersonation allows servers to access data on behalf of privileged clients by assuming the security level of the client. With this arrangement, a user on the network with out the proper security clearance would be denied access to a file through the application server when the server attempts accessing the file by impersonating the unauthorized user. The client's APIs provide the information needed for a thread to take on the security context of the client. The named thread then receives the proper access validation.

Because it is possible that a multi-user server process will service requests from several single-user application processes, it is necessary that the multi-user server process be able to access the correct user global name space when referencing a named object on behalf of some single-user application process. For this purpose, the concept of user impersonation is used and extended so that impersonation not only allows the server process to assume the security context of the client process but also allows it access to the object name space of the client process.

A method for allowing single user applications to operate in a multi-user environment without modification of the single-user application has been described. The method involves modification of the multi-user operating system by creating distinct user name spaces that have a user global context and by extending the concept of impersonation to make the user name spaces available to the application server.

The method described above is further summarized in the flow diagram of Fig. 10 where it is identified as method 400 for creating a coexistent user and system global context. Step 401 establishes a user global context by assigning a label to each instance of an object or application that is to be used by a single user. A single-user name space is thereby created by identifying each such instance as being globally available to the specified single-user. Step 402 enables the server process to impersonate the single-user by assuming its identity and thereby provides the server access to the single-user name space. Step 403 establishes a system global context by adding a system global identifier to each of the executable files and dynamically linkable library files. The method ends with step 404 establishing the user global context as the default context.

As will be understood by those skilled in the art, many changes in the methods described above may be made by the skilled practitioner without departing from the spirit and scope of the invention, which should be limited only as set forth in the claims which follow.

CLAIMS

What is claimed is:

1. A method for allowing a single-user application program to operate in a multi-user operating system environment without modifying the single user program by modifying the operating system methods used for object name creation, look-up, and deletion, for allowing multiple copies of a single-user application to run simultaneously, the method comprising:

(a) assigning a label to each instance of an object and to each instance of an application to be used only by a single-user, for identifying that single-user and for creating a distinct single-user name space by identifying each of such instances as being globally available only to the single-user; and

(b) enabling a server process that is serving the application of the single-user to impersonate the single-user by assuming the identity of the single-user process, thereby allowing the server process to access the single-user name space.

2. The method of claim 1 wherein the step of assigning a label to each instance of an object and each instance of an application, further comprises attaching a single-user identifier to a shared object identifier.

3. The method of claim 2 wherein the shared object is an application.

4. The method of claim 2 wherein the shared object is a section of memory.

5. The method of claim 2 wherein the shared object is a pipe object.

6. The method of claim 2 wherein the shared object is a communication port object.

7. The method of claim 2 wherein the shared object is a synchronization object.

8. The method of claim 7 wherein the synchronization object is an event object.

9. The method of claim 7 wherein the synchronization object is a mutex object.

10. The method of claim 7 wherein the synchronization object is a semaphore object.

11. The method of claim 5 wherein the synchronization object is a pipe object.

12. A method for creating a user global context for allowing a single-user application program to operate in a multi-user operating system environment without modifying the single user program, and for providing a coexisting system global context by modifying the operating system methods used for object name creation, lookup, and deletion, the method comprising:

(a) establishing a user global context by assigning a label to each instance of an object and to each instance of an application that is to be used only by a single-user, for identifying the single-user and for creating a distinct single-user name space by identifying each such instance as being globally available only to the single-user;

(b) enabling a server process that is serving the application of the single-user to impersonate the single-user by assuming the identity of the single-user process, thereby allowing the server process to access the single-user name space; and

(c) establishing a system global context by labeling each system global named resource by adding a system global identifier to each of the system global named resource's associated executable files and dynamically linkable library files.

13. The method of claim 11 further comprising the step of establishing the user global context as a default context.

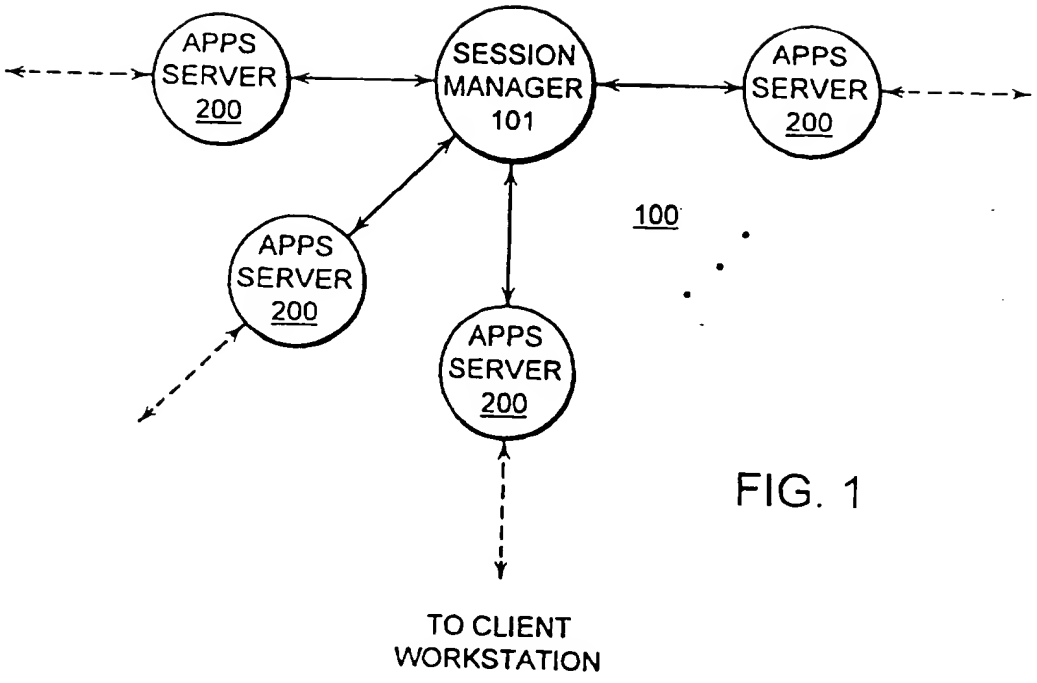
APPS SERVER 200
SESSION MANAGER 101
APPS SERVER 200
100
APPS SERVER 200
APPS SERVER 200
TO CLIENT WORKSTATION

FIG. 1

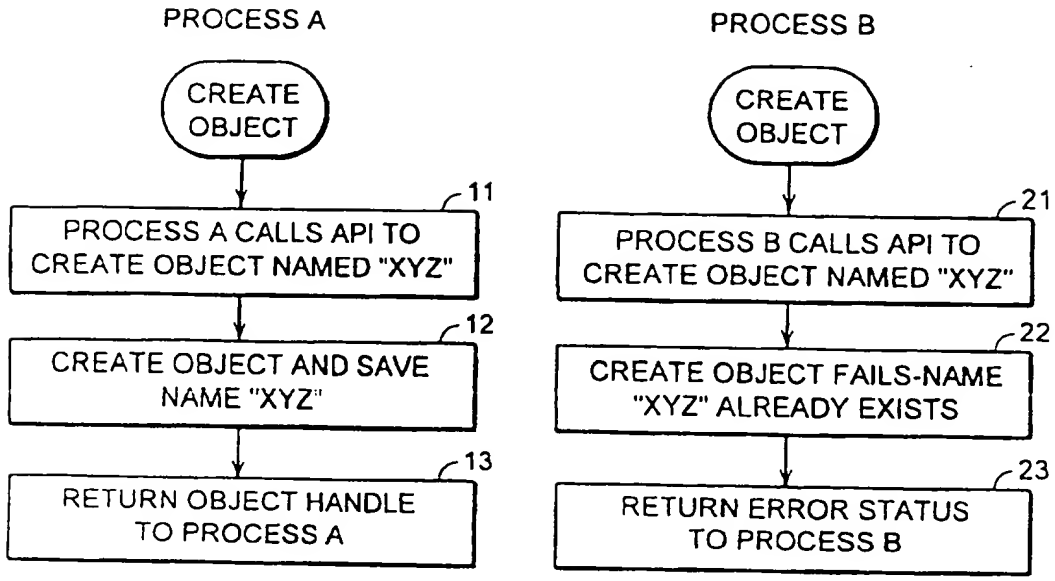

FIG. 3(a) (PRIOR ART)

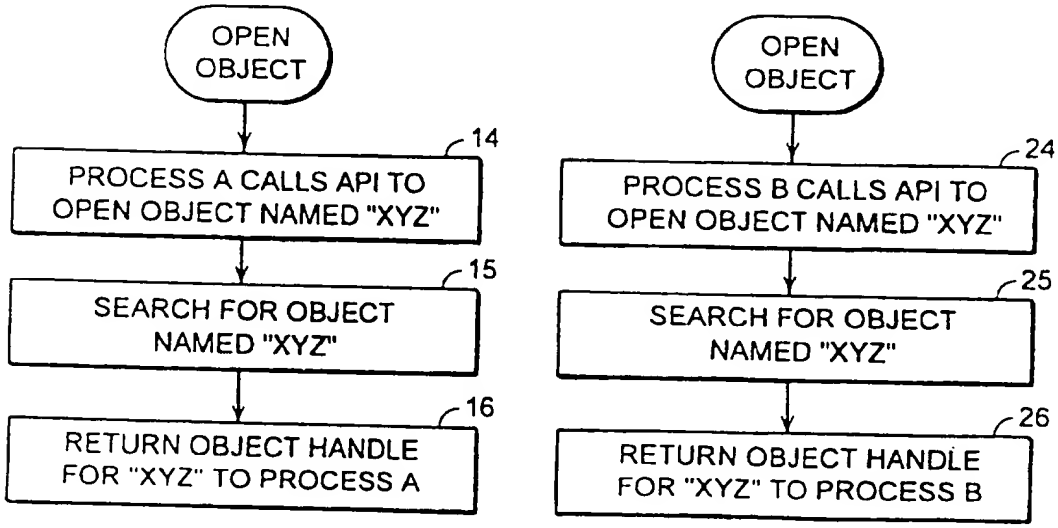

FIG. 3(b) (PRIOR ART)

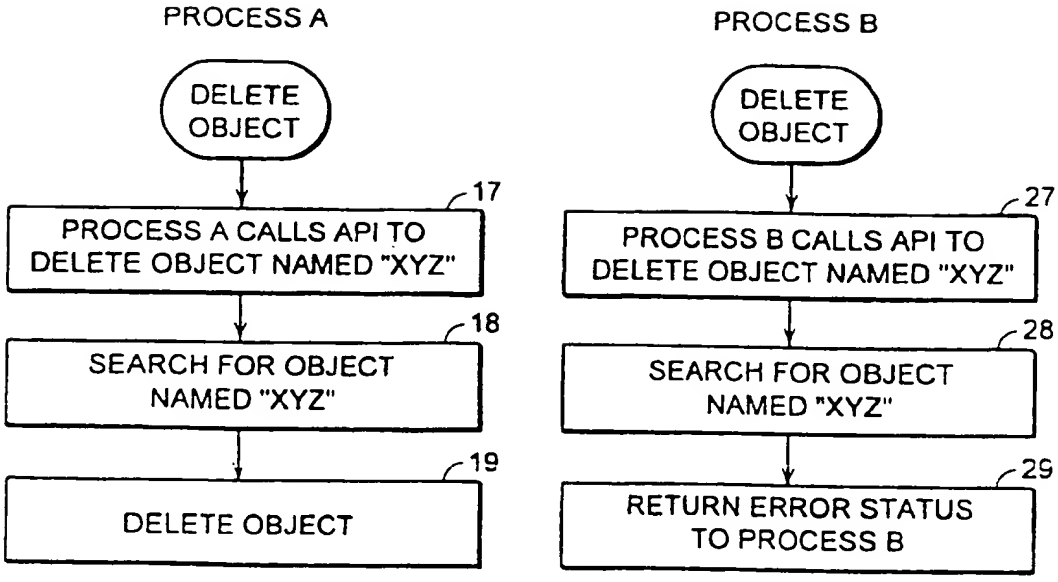

FIG. 3(c) (PRIOR ART)

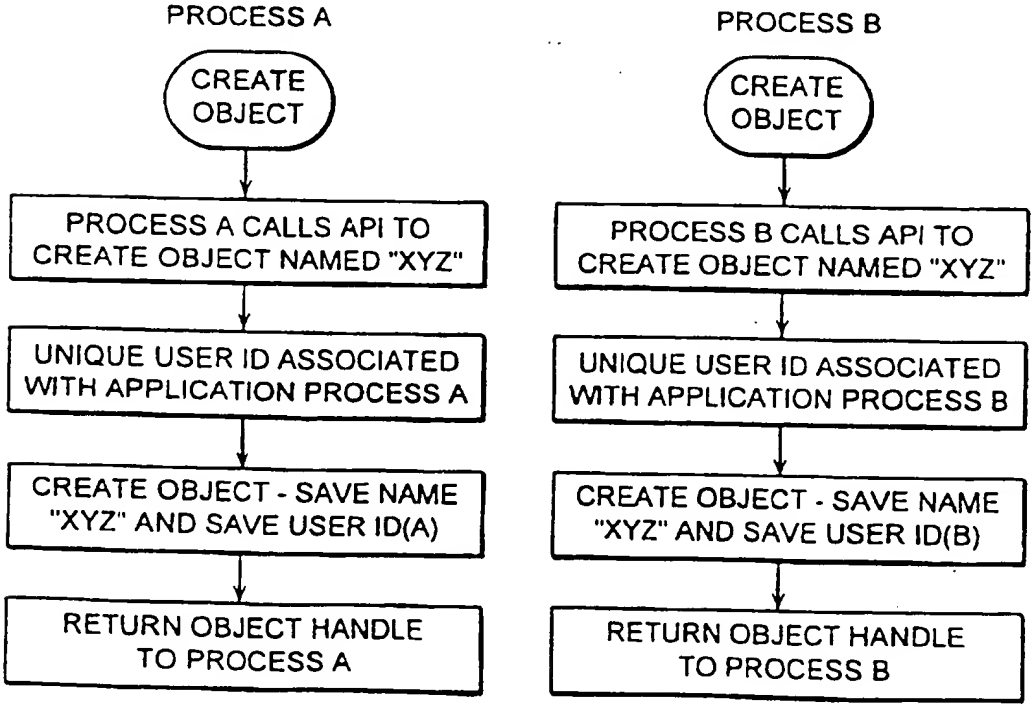

FIG. 4(a)

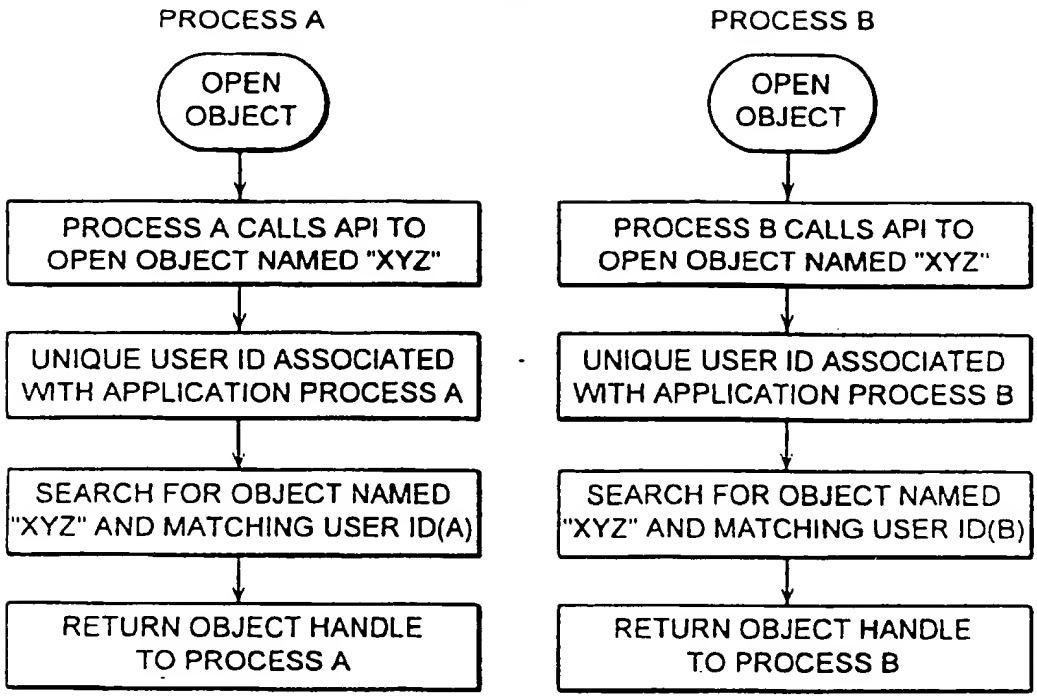
PROCESS A
OPEN OBJECT
PROCESS A CALLS API TO OPEN OBJECT NAMED "XYZ"
UNIQUE USER ID ASSOCIATED WITH APPLICATION PROCESS A
SEARCH FOR OBJECT NAMED "XYZ" AND MATCHING USER ID(A)
RETURN OBJECT HANDLE TO PROCESS A
PROCESS B
OPEN OBJECT
PROCESS B CALLS API TO OPEN OBJECT NAMED "XYZ"
UNIQUE USER ID ASSOCIATED WITH APPLICATION PROCESS B
SEARCH FOR OBJECT NAMED "XYZ" AND MATCHING USER ID(B)
RETURN OBJECT HANDLE TO PROCESS B

FIG. 4(b)

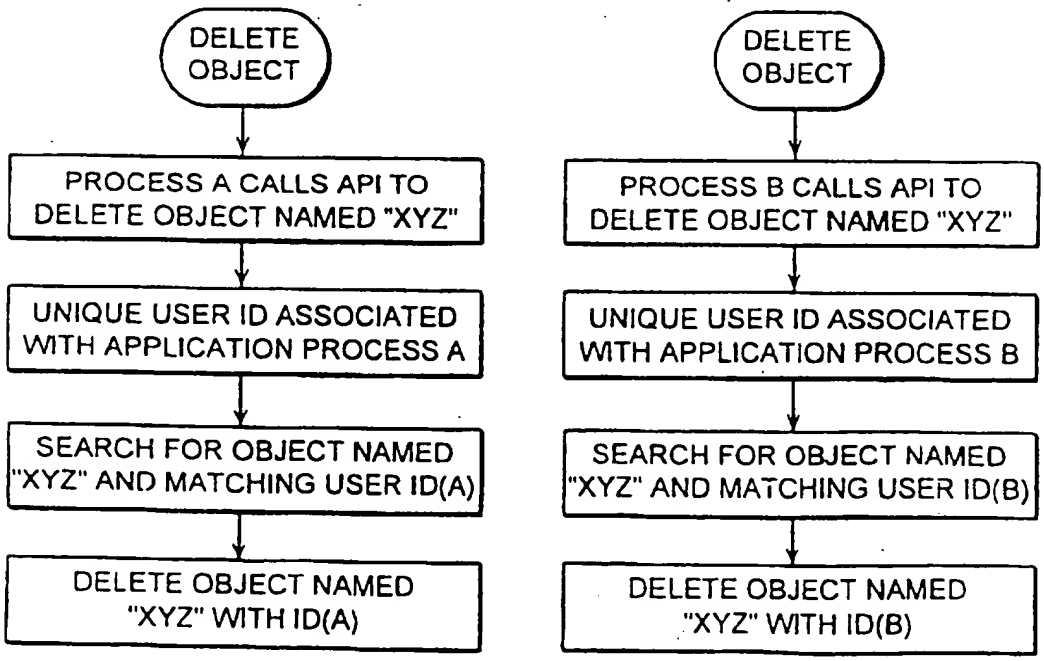
DELETE OBJECT
PROCESS A CALLS API TO DELETE OBJECT NAMED "XYZ"
UNIQUE USER ID ASSOCIATED WITH APPLICATION PROCESS A
SEARCH FOR OBJECT NAMED "XYZ" AND MATCHING USER ID(A)
DELETE OBJECT NAMED "XYZ" WITH ID(A)
DELETE OBJECT
PROCESS B CALLS API TO DELETE OBJECT NAMED "XYZ"
UNIQUE USER ID ASSOCIATED WITH APPLICATION PROCESS B
SEARCH FOR OBJECT NAMED "XYZ" AND MATCHING USER ID(B)
DELETE OBJECT NAMED "XYZ" WITH ID(B)

FIG. 4(c)

	EXE/DLL MARKED SYSTEMGLOBAL	EXE/DLL NOT MARKED SYSTEMGLOBAL
SYSTEMGLOBAL NOT SPECIFIED ON API CALL	SYSTEMGLOBAL	USERGLOBAL
SYSTEMGLOBAL SPECIFIED ON API CALL	SYSTEMGLOBAL	SYSTEMGLOBAL

FIG. 5

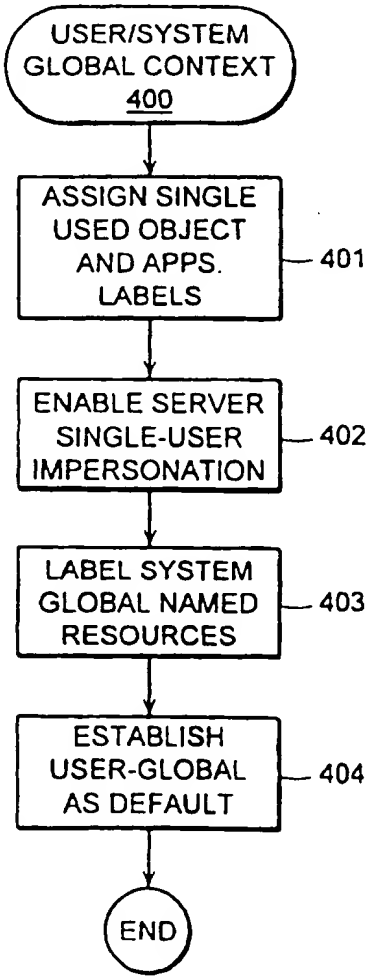

FIG. 6

INTERNATIONAL SEARCH REPORT

Inter ional Application No
PCT/US 96/15947

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F9/455 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	BYTE, vol. 16, no. 1, January 1991, ST PETERBOROUGH US, pages 134-138, XP002023070 JO UDELL: "Citrix's New Multiuser OS/2" see the whole document ---	1,12
A	WO,A,94 14114 (OVERLORD, INC.) 23 June 1994 see page 1, line 3 - page 7, line 19 ---	1,12
A	BYTE, vol. 13, no. 13, December 1988, ST PETERBOROUGH US, pages 183-188, XP002023071 JOHN UNGER: "The Sun386i" see the whole document --- -/--	1,12

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
20 January 1997	06. 02 97
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+ 31-70) 340-2040, Tx. 31 651 epo nl, Fax (+ 31-70) 340-3016	Authorized officer Fonderson, A

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intern· ial Application No
PCT/US 96/15947

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	BYTE, vol. 13, no. 13, December 1988, ST PETERBOROUGH US, pages 207-212, XP002023072 JEFF HOLTZMAN: "Merge 386" see the whole document -----	1,12

1

INTERNATIONAL SEARCH REPORT

information on patent family members

Inter nal Application No
PCT/US 96/15947

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
WO-A-9414114	23-06-94	AU-A-	5680894	04-07-94
		CA-A-	2103297	08-06-94
		CN-A-	1091842	07-09-94
		EP-A-	0679271	02-11-95
		JP-T-	8506912	23-07-96